**Week 5 Assignment**

Guruprasad Velikadu Krishnamoorthy

College of Science and Technology, Bellevue University

[**DSC650**-T301: Big](https://cyberactive.bellevue.edu/webapps/blackboard/execute/courseMain?course_id=_512542_1) Data

Professor. Nasheb Ismaily

December 19, 2023

1. **SparkSQL with Scala:**

val grades\_df\_scala = spark.read.format("csv").option("header", "true").load("/data/grades.csv")

grades\_df\_scala.createOrReplaceTempView("grades\_df\_sql")

spark.sql("SHOW TABLES").show()

spark.sql("SELECT \* FROM grades\_df\_sql WHERE Final > 50").show()

spark.sql("SELECT \* FROM grades\_df\_sql").show()

A screenshot of a computer

Description automatically generated

**Additional SparkSQL commands used:**

* 1. spark.sql("SELECT \* from grades\_df\_sql order by Final desc;").show()

**Significance:** This command lists the in the Descending order of the values of column “Final”

* 1. spark.sql("SELECT \* from grades\_df\_sql where Test1>=0 and Test2>=0 and Test3>=0 and Test4 >=0 and Final>=0").show()

**Significance:** This returns the records where the Test1,Test2, Test3 ,Test4 and Final are only positive values.

* 1. spark.sql("SELECT AVG(Test1),min(Test1),max(Test1),std(Test1) from grades\_df\_sql ").show()

**Significance:** Returns the statistical quantities of Test1 values.

**A screenshot of a computer

Description automatically generated**

1. **SparkSQL with Python:**

grades\_df\_python = spark.read.format('csv').option('header', 'true').load('/data/grades.csv')

grades\_df\_python.show()

grades\_df\_python.createOrReplaceTempView('grades\_df\_py\_sql')

spark.sql('SHOW TABLES').show()

spark.sql('SELECT \* FROM grades\_df\_py\_sql WHERE Final > 50').show()

A screenshot of a computer

Description automatically generated

**Additional SparkSQL commands used:**

* 1. spark.sql("SELECT \* from grades\_df\_py\_sql order by Final desc;").show()

**Significance:** This command lists the in the Descending order of the values of column “Final”

* 1. spark.sql("SELECT \* from grades\_df\_py\_sql where Test1>=0 and Test2>=0 and Test3>=0 and Test4 >=0 and Final>=0").show()

**Significance:** This returns the records where the Test1,Test2, Test3 ,Test4 and Final are only positive values.

* 1. spark.sql("SELECT AVG(Test1),min(Test1),max(Test1),std(Test1) from grades\_df\_py\_sql ").show()

**Significance:** Returns the statistical quantities of Test1 values.

* 1. spark.sql("SELECT Grade,count(\*) from grades\_df\_py\_sql group by Grade order by Grade").show()

**Significance:** Returns the count of the number of students grouped by the Grade.

* 1. spark.sql("SELECT \* from grades\_df\_py\_sql where (Test1+Test2+Test3+Test4)/4 < Final ").show()

**Significance:** Returns the records where the average of Test1-4 scores is lesser than the Final scores.

* 1. spark.sql("SELECT \*, DENSE\_RANK() OVER (ORDER BY Grade) AS ROW\_RANK FROM grades\_df\_py\_sql").show()

**Significance:** This assigns Rank to the students ordered based on the Grades.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **SparkSQL with Custom Data Set:**

For this exercise, the Hollywood movies dataset used in the prior Hive assignment is reused.

* 1. **The description of some of the commonly used columns is explained below:**
* ***Movie:*** Name of the movie
* ***LeadStudio:*** TheStudio that produced the movie.
* ***RottenTomatoes:*** Represents the score rated by Rotten Tomatoes critics.
* ***AudienceScore***: Represents the score rated by Audience critics.
* ***Genre***: Represents the Genre of the movie
* ***TheatersOpenWeek***: Number of theatres where the movie was released in the opening week in the US and Canada. This is an integer value.
* ***OpeningWeekend*:** Box Office collection in the opening week in the US and Canada in USD. This is a floating number value.
* ***DomesticGross:*** *Total domestic collection in the US and Canada represented in USD.*
* ***ForeignGross:*** *Total collection from countries other than the US and Canada represented in USD.*
* ***WorldGross:*** Sum of Domestic and Foreign Gross collection represented in USD.
* ***Budget:*** Total movie budget represented in USD.
* ***Profitability:*** Total profits made from the movie represented in Percentage.
* ***Year***: The year in which the movie was released.
  1. Commands to copy the dataset into HDFS, start Spark shell and on **Scala**, and load the data into Data frame.

*# Copying file to HDFS*

hdfs dfs -put /data/HollywoodMovies.csv /data/HollywoodMovies.csv

hdfs dfs -ls /data/HollywoodMovies.csv

*# Starting Spark shell on Scala*

spark-shell

*# Creating a Dataframe using Spark Scala and printing sample rows:*

val movies\_df\_scala=spark.read.format("csv").option("header","true").load("/data/HollywoodMovies.csv")

movies\_df\_scala.show(4)

*# Creating a View on the Dataframe to use Spark Sql:*

movies\_df\_scala.createOrReplaceTempView("movies\_df\_sql")

spark.sql("show tables").show()

A screenshot of a computer

Description automatically generated

* 1. **Additional Scala Sql Commands used:**
     1. spark.sql("select count(\*) from movies\_df\_sql").show()

**Significance:** Returns the count of number of records in the Dataframe.

* + 1. spark.sql("select Year,count(Movie) as movie\_count from movies\_df\_sql where Year >=2007 group by Year Order by Year").show()

**Significance:** The query returns the count of the number of movies from each year in the dataframe.

* + 1. spark.sql("select max(Profitability),Year from movies\_df\_sql where Year >=2007 group by Year Order by Year").show()

**Significance:** Returns the maximum profitability achieved by a movie over the period of 2007-2013.

* + 1. spark.sql("select Genre,Year,avg(OpenProfit) as profits from movies\_df\_sql where Year in (2007,2009) and Year is not Null and Genre is not Null group by Genre,Year order by Year,profits").show()

**Significance:** Returns the average Profits made in the years 2007 and 2009 by Genre, the results are ordered in ascending order of Year and average profit.

* + 1. spark.sql("select round(avg(DomesticGross),2) as Avg\_DomesticGross,round(var\_pop(DomesticGross),2) as Var\_DomesticGross,round(stddev\_pop(DomesticGross),2) as Std\_DomesticGross,Year from movies\_df\_sql where Year >=2007 group by Year order by Year").show()

**Significance:** Returns the statistical quantities such as Mean, Variance, and standard deviation of Domestic Gross , arranged by Year.

* + 1. spark.sql("SELECT Movie,Year,Genre,Profitability as `Profit%` from movies\_df\_sql where Profitability in (select max(Profitability) from movies\_df\_sql where Year >=2007 group by Year) order by Year").show(false)

**Significance:** Using Subqueries, the SQL returns the movie name and its Genre with the highest profit % for each year.

A screenshot of a computer

Description automatically generated

* + 1. spark.sql("SELECT Movie,RottenTomatoes,Year, RANK() OVER (PARTITION BY Year ORDER BY RottenTomatoes DESC) AS ROWNUM FROM movies\_df\_sql").show(false)

**Significance:** Ranks the Movies based on the Rotten Tomatoes rating for each year.

A computer screen with a black background

Description automatically generated

* + 1. spark.sql("SELECT \*, round(PERCENT\_RANK() OVER (ORDER BY OpenProfit),2) AS Percentile from (SELECT Movie,OpenProfit,Year from movies\_df\_sql where Year=2010 and Genre='Action' and OpenProfit is not null)").show(50, false)

**Significance:**  Returns the percentile value of Action movies of 2010, based on the Profits made.

A screen shot of a computer

Description automatically generated